Infusing Reliability Techniques into Software Safety Analysis

Ying Shi, NASA Goddard Space Flight Center

Key Words: Software Safety, FMEA, Hazard Analysis, FTA

SUMMARY & CONCLUSIONS

Software safety analysis for a large software intensive system is always a challenge. Software safety practitioners need to ensure that software related hazards are completely identified, controlled, and tracked. This paper discusses in detail how to incorporate the traditional reliability techniques into the entire software safety analysis process. In addition, this paper addresses how information can be effectively shared between the various practitioners involved in the software safety analyses. The author has successfully applied the approach to several aerospace applications. Examples are provided to illustrate the key steps of the proposed approach.

1 INTRODUCTION

The main objectives of software safety analysis is to identify hazards that are caused or controlled by software, to establish a process to eliminate or mitigate the identified hazards, and to develop methods to verify the safety controls. Software safety analysis for a large software intensive system is always a challenge. Software safety practitioners need to ensure that software related hazards are completely identified, controlled, and tracked. First, all the system level hazards have to be identified. This could be accomplished by starting with the generic hazard list and continuing to system specific hazards. System level hazards are normally identified through brainstorm sessions or based on the inputs from experienced systems engineers. Can this approach capture all the system level hazards? Even if all the system level hazards have been completely identified, how do you ensure that software related hazardous events are correctly recognized? The involvements and contributions of software to hazards are typically more difficult to identify and track. The interaction between the entire system and software functions, as well as between software functions, needs to be clearly understood. Furthermore, all the possible faulty states of the identified software need to be analyzed for their potential impacts on the system, as well as the likelihood and severity of the impacts. Corresponding controls can then be proposed to eliminate or mitigate the identified software hazards and therefore ensure the safety of the software and the system. These tasks will not seem new to reliability engineers. Indeed, these software safety analysis objectives and requirements should sound very familiar to reliability engineers. For instance, reliability engineers use Failure Modes and Effects Analysis (FMEA) which involves reviewing as many components, assemblies, and subsystems as possible to identify failure modes, and their causes and effects.

Researchers and practitioners have introduced reliability techniques, such as FMEA and Fault Tree Analysis (FTA), to software safety analysis [1] [2] [3]. As addressed in [1], each technique has its strengths, as well as limitations. FMEA is normally very time-consuming if applied to all parts of a complex design. FTA will not be able to model transitions and timing related events. How to use these existing techniques properly and effectively in the entire system development life cycle remains unclear. This paper will first briefly review the current available software safety techniques and then discuss how to effectively infuse reliability techniques into the software safety analysis, from hazards identification to hazard cause and control analysis.

The remainder of this paper is organized as follows: Section 2 provides a brief review of the existing software safety techniques; Section 3 highlights techniques for hazards identification, hazards control, and tracking; Section 4 addresses the importance of collaboration, and Section 5 provides a summary of applying reliability techniques throughout the system development lifecycle.

2 REVIEW OF SOFTWARE SAFETY TECHNIQUES

Many software safety analyses are still conducted in an ad hoc manner. Basic techniques such as brainstorming, checklists, and experiences from previous projects are normally used to facilitate the software safety analysis. It is good to start with these analyses as they are easy to execute, but it is difficult to ensure their completeness.

More systematic techniques, such as Hazard and Operability Study (HAZOP) which have been piloted in the chemical industry, have not been widely used in large aerospace applications. This is because of its labor-intensive nature and sometimes HAZOP has to rely on subjective judgments.

Other advanced modeling techniques, such as Dynamic Flowgraph Method (DFM) [4] or Petri-Nets [5] require accurate system modeling and therefore are very complicated for large scale applications. Furthermore, they cannot be applied early in the development lifecycle.

Failure Modes and Effect Analysis (FMEA) is a “bottom up” inductive failure analysis technique. It starts with the failure of one component and its possible failure modes. For each failure mode, how the failure propagates through the system is determined. Next the likelihood and severity of the effects are evaluated. Fault tree analysis (FTA) is a “top

down" deductive failure analysis. An undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events. Event Tree analysis (ETA) uses a forward search to identify all possible consequences of a giving initiating event. These analyses can also be very time consuming to execute, but if used properly and systematically for hazard analysis purposes as shown later in section 3, they can be very effective in the hazard identification and control process.

3 SOFTWARE SAFETY ANALYSIS

Software safety analysis is associated with system hazards analysis and starts with hazards identification. Software's involvement in such hazards will need to be investigated. If software is determined to be the cause of a hazard or is involved in controlling a hazard, the software will be labelled as safety critical software. For NASA projects, the software safety standard [6] will be applicable for such safety critical software.

3.1 Hazard Identifications

At the early project life cycle, system level hazards will be identified first. This could be accomplished by starting with generic hazard lists. The generic hazards list for the space shuttle [2], shown below as Table 1, is used as a general guideline at NASA for system level hazard identification. As seen from Table 1, most of the generic hazards are physical hazards and are not caused by software. Software, however, can be used to control certain hazards. For instance, operational or surviving heaters are used to mitigate extreme cold conditions for satellites in space. Software is involved in controlling the proper operations of such heaters and therefore could be safety critical.

System specific hazards will then need to be identified. We want to understand the scenario and consequence of a specific initiating event at a high level. Basic techniques such as brainstorming, checklists, and experiences from previous projects can be used. But to ensure the completeness of the hazard identification, we advocate functional FMEA (FFMEA). FFMEA is based on a functional breakdown of a system and starts with each of the system breakdowns. It can also be done early in the life cycle phase, i.e., as soon as the preliminary system architecture is available.

In FFMEA, functions can be evaluated on potential functional failure effects before design details are available. The safety impact will then be assessed based on the evaluated failure effects. If an impact on system/personal safety is assessed, the function will be identified as a system hazard. General controls or mitigations can be proposed to limit consequence of functional failures or limit the probability of occurrence in this early development. For example, if a system decides to go with a deployable instrument cover, one needs to evaluate the possible safety impact of an unsuccessful cover deployment or an inadvertent cover deployment during system integration testing, during launch and orbit ascent, and during the normal operational mission phase.

Table 1 Generic Hazards Checklist

Hazard Category	Hazards
Contamination/ Corrosion	Chemical Disassociation; Chemical Replacement/Combination; Moisture; Oxidation; Organic; Particulate; Inorganic
Electrical Discharge/Shock	External/Internal Shock; Static Discharge; Corona; Short
Environmental/ Weather	Fog; Lightning; Precipitation; Sand/Dust; Vacuum; Wind; Temperature Extremes
Fire/Explosion	Chemical Change; Fuel & Oxidizer in Presence of Pressure and Ignition Source; Pressure Release/Implosion; High Heat Source
Impact/Collision	Acceleration; Detached Equipment; Mechanical Shock/Vibration/Acoustical; Meteoroids/Meteorites; Moving/Rotating Equipment
Loss of Habitable Environment	Contamination; High Pressure; Low Oxygen Content; Low Pressure; Toxicity; Low Temperature; High Temperature
Pathological/ Physiological/ Psychological	Acceleration/Shock/Impact/Vibration; Atmospheric Pressure; Humidity; Illness; Noise; Sharp Edges; Lack of Sleep; Visibility; Temperature; Workload Excessive
Radiation	EMI; Ionizing Radiation; Non-ionizing Radiation
Temperature Extremes	High, Low, Variations

Using FFMEA and by following the functional breakdown of a system could ensure all the subsystems or functions are covered in the analysis and their impacts on system safety have been evaluated. An example worksheet as shown below in Table 2 can be created to document FFMEA results. Example spacecraft functions that could have safety impacts and therefore result in system hazards are listed below:

- Deployment of solar array
- Deployment of instrument cover
- Deployment of antenna
- Deployment of Magnetometer boom
- Laser firing
- Thrusters firing

Table 2 Example FFMEA Worksheet

Functions	Functional Failure Mode	Failure Effects	Safety Impact	Impact Phases
F_1: Instrument Cover Deployment	F_1M_1: Inadvertent Deployment	Cover damage and spacecraft damage due to cover debris	Yes	Ground Integration and Testing, Launch and Orbit Ascent
	F_1M_2: Can not deploy	Instrument can not work and mission failure	Yes	Normal Mission Operation

3.2 Role of Software in Hazards

Once system hazards have been clearly identified, software's contribution to the system hazards needs to be investigated. In many cases, an experienced systems engineer will be able to identify whether there is software associated with an identified system hazard. But this experience-based assessment is not always precise and complete. Furthermore, software's involvement in certain hazard scenarios is not obvious. Scenario-based reliability analysis technique, e.g., Fault tree analysis and Event tree analysis can be used to assist in identifying software involvement and contribution to a certain hazard. A Fault tree starts with an undesirable top event, in this case a system hazard, and breaks down to different levels of trees. Each level consists of more basic events that are necessary and sufficient to cause the event shown in the level above it. This approach is very effective in identifying software's contribution in complex system design. We are not intended to quantify a specific system hazard, a qualitative fault tree is sufficient in this step. Developing a fault tree down to the software interface level is our major objective. Software interfaces with other system components will be clearly illustrated in the fault tree. Whether software will be a cause or control of a hazard will be clearly illustrated, as well.

We will continue to explain this step using a simplified inadvertent instrument cover deployment design as an example. The actuation of the deployment is implemented by an actuator. Typically inhibits, either hardware or software or a combination of both, are provided to prevent inadvertent deployment. The actuators have independent spacecraft-provided power services. An arming plug is in line with actuator power and is installed only for critical ground integration and test (I&T) activities and for launch. To actuate an instrument cover, the spacecraft will send an independent command through flight software to turn on the instrument power and specific actuator interface. Next the spacecraft will coordinate with system hardware (e.g., FPGA) to enable the commanding process and coordinate with system software (flight software) to initiate independent two-step commands. This is used to first arm the actuator and then fire the actuator.

At a first glance of the above design description, commands sent through flight software are involved in the entire actuation process. One may not identify flight software (FSW) as a system inhibit in the actuation process until the following fault tree (Figure 1) is developed and the software interfaces, as well as the interactions with other system components, are clearly shown. FSW is involved in all three "and" gates and has to work with other system hardware. For example, FSW will send the arm command to the actuator to arm the instrument cover. However, this command will not be valid without a system hardware enable signal. Before launch (except for ground I&T), the arming plug, Instrument power OFF, actuator arm sequence ("and" gates #2) and actuator fire sequence ("and" gate #3) are considered four system inhibits for this hazard. The malfunction of FSW will take out two of the inhibits (instrument power and "and" gate #3) and the remaining two inhibits are still sufficient to meet the system safety requirement (i.e., two inhibits are required for critical system hazards). After launch and during I&T, the arming plug is installed and the instrument is turned on. Only "and" gate #2 and #3 are active inhibits. If the enable signals are purely hardware controlled, FSW will not be considered as safety critical software. From the fault tree, however, we found out that the hardware switch for actuator fire sequence is also controlled by FSW. If there is a bug in the FSW, it will take out "and" gate #3 entirely leaving only one system inhibit. Therefore FSW functions involved in this hazard control are critical and will be considered as safety critical software functions.

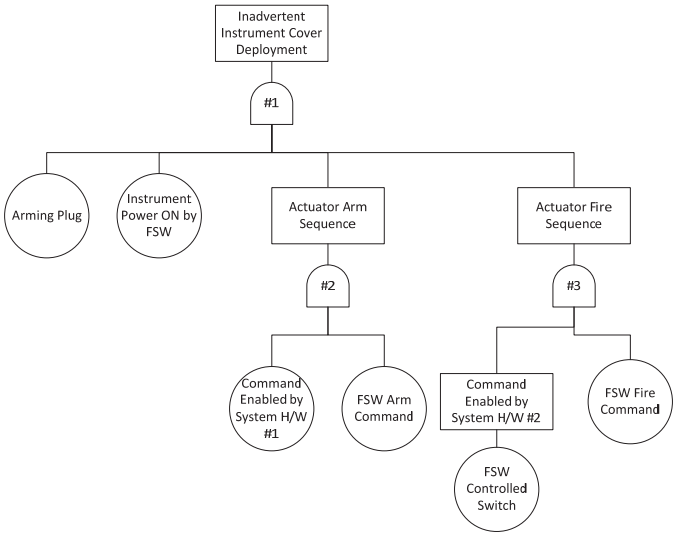

Figure 1 Example Fault Tree for Inadvertent Instrument Cover Deployment

3.3 Hazard Control

As discussed in section 3.1, functional FMEA will be used early in the development lifecycle to analyze the system at a high level. As soon as the specific safety critical software function are identified, detailed design FMEA can be implemented for each specific critical software function for their associated failure modes, causes and effects. The effects are described in terms of their potential impact to personnel or mission success and are evaluated for the likelihood of occurrence and severity of potential impact. Since in this step, FMEA is applied mainly on software, we can call it Software FMEA (SFMEA). At the preliminary design review (PDR), software functions in the form of Computer Software Configuration Items (CSCIs) or Computer Software Units (CSUs) are the target items. Later in the life cycle, e.g., critical design phase (CDR), lower levels of the breakdowns, such as modules, may be used to start the SFMEA process.

By continuing the inadvertent deployment of instrument cover example, flight software commanding functions are investigated. Three failure modes may be identified by PDR:

- Inadvertent commanding,
- Failure to command and

- Incorrect commanding.

More specific failure modes will need to be evaluated by CDR. These failure modes are specific to detailed design, from the module level down to the line of code level. Each function failure mode will be decomposed to component level. Detailed SFMEA guidance and instructions can be found in many literatures, e.g. [2] [9] [10]. Example failure mode categories include:

- Logic/Algorithm
- Data
- Timing/Sequence
- Hardware
- Memory
- Interface

As for likelihood and severity, one can follow the guidance in MIL-STD-882 [9] as shown in Table 3 and Table 4.

It should be noted that by applying SFMEA only on an identified critical software function, one can obtain a comprehensive analysis on this particular function. This focused approach will avoid unnecessary analysis on other non-critical functions.

Once the failure mode and cause with severity and likelihood of the end effects have been laid out, hazard controls can then take the form of eliminating hazards, or mitigating hazards by reducing the likelihood of a hazard occurring, by minimizing the severity or by improving the awareness and notification of a hazardous state or condition. The verification approach includes, but is not limited to analysis, inspection, demonstration, and test.

Table 3 Severity Categories

Severity	Mission Effects
Catastrophic	Death, permanent total disability, irreversible significant environmental impact, loss ≥ $10M
Critical	Permanent partial disability, severe injuries or occupational illness, reversible significant environmental impact, or $1M ≤ loss ≤ $10M
Marginal	Moderate injury or occupational illness, reversible moderate environmental impact, or monetary loss equal to or exceeding $100K ≤ loss ≤ $1M
Negligible	Minor injury or occupational illness, minimal environmental impact, loss ≤ $100K

Table 4 Likelihood Levels

Level	Likelihood
Frequent	Likely to occur often in the life of an item
Probable	Will occur several times in the life of an item
Occasional	likely to occur sometime in the life of an item
Remote	Unlikely, but possible to occur in the life of an item
Improbable	So unlikely, it can be assumed occurrence may not be experienced in the life of an item

All the analysis results need to be documented in the SFMEA worksheet, similar to the example shown in Table 5.

Table 5 Example SFMEA Worksheet

Software Component	Related Requirements	Failure Mode and Cause	Failure Effects	Severity	Likelihood	Hazard Controls	Verification Method

3.4 Hazard Tracking

Hazards with any software safety issues need to be tracked to closure, either by software assurance or safety assurance practitioners. Existing project tracking systems can be used to track software safety issues.

In addition, many software related failures/accidents are due to requirement flaws (i.e., missing or incorrect requirements). Requirements analysis plays a critical role in the software safety analysis. The identification of safety critical requirements starts by analyzing the decomposition from the system safety requirements. Later, as described in section 3.2, once critical software/software functions are identified, one needs to make sure related requirements are marked as safety critical.

Software safety requirements, as a subset of the functional requirements, will need to be correct, complete, consistent and verifiable. Existing software engineering and assurance standard and guidance, e.g. [7] [8], are used to ensure the quality of the requirements. As for safety critical requirements, we will need to ensure they are marked and can be tracked throughout the development life cycle. By using the fault tree analysis shown in 3.2, one needs to cross check that all the safety critical software identified by a fault tree with their requirements are clearly marked in the requirements database. Once there is any change to such safety critical requirements, its impact on system safety will then need to be re-evaluated. Existing requirements databases, e.g., DOORS, can be used to store and track safety critical requirements.

4 COLLABRATIONS

The project's software safety engineer (SSE) is the primary person responsible for software safety analysis. The SSE must be familiar with the system and subsystem level architecture and requirements, must understand the nature of the system and software hazards, and must be able to develop fault trees and FMEAs for hazard analysis. But since no single person understands all system components, software or hardware, it is important to interact and collaborate with other team members to effectively understand the full scope of software and to develop fault trees and software failure modes and effects analysis. Critical information needs to be shared among software engineers, systems engineers, and safety and reliability representatives. For example, as you are performing your software safety analysis, have your hardware and software systems engineers review your system decomposition analysis. Request that your reliability engineer review your fault tree and FMEAs and have your software

designers review your software requirements analysis.

We still need some formalized meeting or review board to review and approve the analysis results. The reviewers point of view will help uncover hidden assumptions or identify analysis insufficiency. At NASA, a system safety review board (SSRB) ensures that all hazards identified by the system and software safety team are adjudicated by program management and customer representatives. The SSRB also ensures that hazards are properly tracked throughout the development life cycle and that hazard risks are properly communicated to stakeholders.

5 SUMMARY

A summary of the proposed approach for infusing reliability techniques in software safety analysis can be illustrated using the flowchart shown in Figure 2. One can start with the generic hazards checklist for generic hazards. A Functional FMEA can help identify system specific hazards. Once system hazards are clearly identified, software involvement, either as a hazard cause or control can be identified using fault tree analysis. Safety critical software functions identified by a fault tree will then be analyzed in detail using the Software FMEA process.

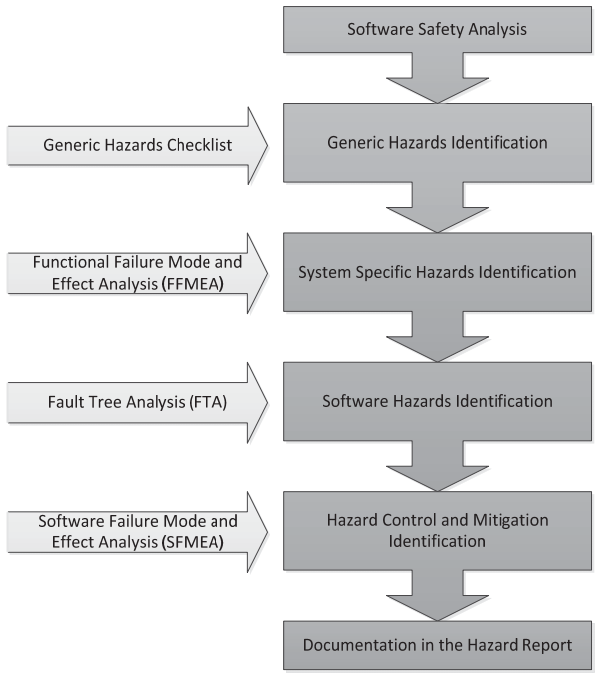

Figure 2 Infusing Reliability Techniques into Software Safety Analysis

It should be noted that brainstorming and checklists are still actively involved in the entire approach. However, by following the FMEA and FTA process, one can assure the completeness of the failure/hazard analysis. Also by applying SFMEA only on identified critical software functions one can avoid unnecessary analysis on other non-critical functions. These are the two major strengths of using reliability techniques in software safety analysis.

REFERENCES

[1] N. Leveson, Safeware:System Safety and Computers, Addison-Wesley, 1995.

[2] NASA, "NASA Software Safety Guidebook NASA-GB-8719.13," Washington DC, 2004.

[3] Joint Services Computer Resources Management Group, U.S. Navy,U.S. Army, and the U.S. Air Force, "Software System Safety Handbook: A Technical & Managerial Team Approach," 1999.

[4] Garrett, C., Guarro, S., Apostolakis, G., "The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems," *IEEE Trans. on Systems, Man and Cybernetics.*, vol. 25, pp. 824-840, 1995.

[5] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, pp. 541-580, 1989.

[6] NASA, "NASA Software Safety Standard NASA-STD 8719.13C," Washington DC, 2013.

[7] Herbert Hecht, Xuegao An,Myron Hecht, "Computer Aided Software FMEA for Unified Modeling Language," in *RAMS*, 2004.

[8] N. W. Ozarin, "Applying Software Failure Modes and Effects Analysis to Interfaces," in *RAMS*, 2009.

[9] DoD, "Department of Defense Standard Practice System Safety MIL-STD-882E," 2012.

[10] NASA , "NASA Software Engineering Requirements, NASA-NPR-7150.2A," Washington DC, 2009.

[11] DoD, "Software Considerations in Airborne Systems and Equipment Certification, DO-178B," 1992.

BIOGRAPHIES

Ying Shi, PhD
NASA Goddard Space Flight Center, Code 320.1
Greenbelt, MD 20771

Email: ying.shi@nasa.gov

Dr. Ying Shi is a Software Reliability and Safety Engineer at NASA Goddard Space Flight Center, providing technical support for all Goddard projects related to software reliability and safety. She received her Ph.D. in Software Reliability Engineering from the University of Maryland and her M.S. in Reliability Engineering from the University of Arizona. She is a member of IEEE and IEEE Reliability Society.